Der Apex Session State

Jürgen Sieben

Der Session State ist eine zentrale Komponente jeder Web-Anwendung. Nur durch ihn ist es möglich, seitenübergreifend Informationen für eine Anwendung bereitzustellen.

HTTP ist bekanntermaßen ein zustandsloses Protokoll, sodass nur durch einen externen Session State in Verbindung mit einer Session-ID, die durch die Anwendung verwaltet wird, dieses Protokoll überhaupt zur Anwendungsentwicklung genutzt werden kann. Apex implementiert den Session State - naheliegenderweise als Tabelle des Benutzers „Apex_nnnnnn“. Der Zugriff auf den Session State ist daher leicht aus SQL und PL/SQL möglich. Dennoch sind der aktuelle Zustand des Session State und die verschiedenen Möglichkeiten des Zugriffs auf ihn ein Thema, das Einsteiger in Apex regelmäßig zur Verzweiflung bringt. Der Artikel bringt Licht ins Dunkel.

Der Session State ist ganz trivial eine Tabelle im Schema „Apex_nnnnnn“ mit dem Namen „WWV_FLOW_DATA“. Darin wird für jede Session (die man an ihrer Session-ID erkennt und die in dieser Tabelle „FLOW_INSTANCE“ heißt) und für jedes Element der Anwendung der aktuell gültige Wert hinterlegt. Eine Ausnahme bilden die Kollektionen, die innerhalb von Apex verwendet werden, um zum Beispiel Tabular Forms im Session State zu halten. Diese sind in der Tabelle „WWV_FLOW_COLLECTIONS$“ gespeichert.

Wozu benötigen wir den Session State eigentlich? Denn wenn der Session State letztlich eine einfache Tabelle ist und die Daten der Anwendung ja ohnehin aus Tabellen des Workspace-Schemas kommen, wozu sind diese dann in einer Tabellen-Landschaft von Apex gespeichert und nicht direkt in den Tabellen, aus denen sie kommen? Der Grund liegt in der Implementierung der Transaktionen in Apex.

Ein Seitenaufruf (ohne Zwischenwege wie dynamische Aktionen etc.) besteht aus einer Anfrage und einer Antwort, also „Request“ und „Response“. Interessant ist hier insbesondere der Request, also die Anfrage eines Benutzers, denn in dieser Anfrage sind häufig Daten enthalten, die in der Datenbank gespeichert werden müssen.

Ein Request ist immer eine Transaktion, sie wird also im Erfolgsfall mit „commit“ abgeschlossen. Dies muss so sein, denn Apex verfügt über einen Connection Pool (genauer: ORDS verfügt über einen solchen Pool), sodass nicht sichergestellt ist, dass ein in der gleichen Apex-Anwendung folgender Request des gleichen Benutzers über die gleiche Da-

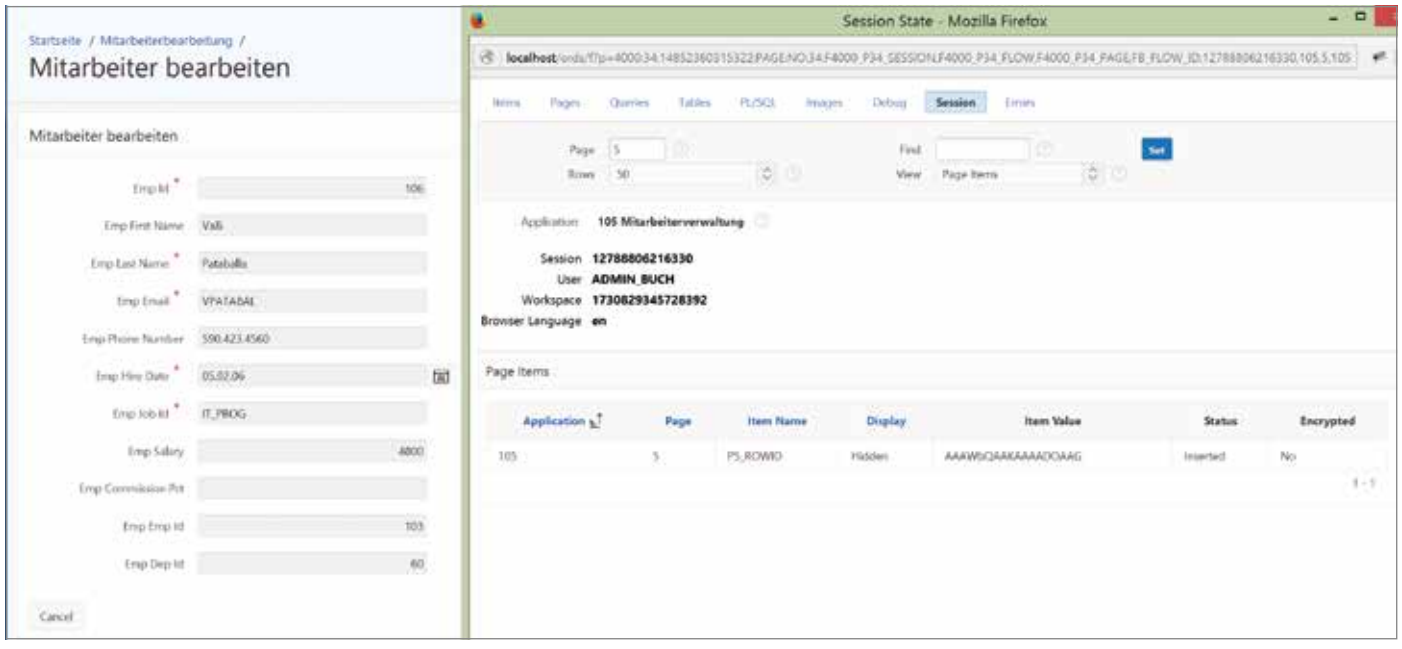

Abbildung 1: Der Session State ist leer

tenbank-Verbindung abgewickelt wird. Eine Transaktion kann also technisch nur in einem Request vereinbart werden und nicht mehrere Requests umfassen.

Ist auf der Seite, die den Request stellt, ein Formular enthalten (bei Apex ist das immer so), werden alle Formular-Elemente der Seite an Apex gesendet. Der Prozess der Verarbeitung dieses Request ist komplex und umfasst eine Reihe von Einzelschritten, die jeder für sich auch scheitern könnten. Ist dies irgendwann der Fall, muss Apex in der Lage sein, die Seite mit den Eingaben des Benutzers erneut aufzubauen, um ihm die Möglichkeit zur Fehlerkorrektur zu geben. Natürlich sollten diese fehlerhaften Daten sich dann noch nicht in den eigentlichen Workspace-Tabellen befinden (je nach Fehler können die Daten auch gar nicht in die Tabellen eingefügt werden, weil zum Beispiel ein „Not Null"-Constraint verletzt sein könnte), sondern irgendwo anders, und das ist eben im Session State.

Ein weiteres Anwendungsszenario ist ein Assistent, der über mehrere Seiten hinweg Daten zusammenträgt, um abschließend eine oder mehrere Tabellen des Workspace-Schemas damit zu füllen. Jede einzelne Seite des Assistenten ist ein abgeschlossener Request-Response-Zyklus, der alle eingegebenen Daten speichern, die Ursprungstabellen aber nicht ändern soll, bis schließlich der Assistent als Ganzes bestätigt oder auch verworfen wird. Auch dies wäre ohne einen generischen Session State nicht möglich. Letztlich gibt es auch Daten, die nicht in Tabellen des Workspace-Schemas gespeichert, aber in der Apex-Anwendung benötigt werden, etwa der Name des angemeldeten Benutzers oder andere Angaben. Auch hierfür steht der generische Session State zur Verfügung.

Den Session State beschreiben

Der Session State in der Datenbank kann (und wird auch sehr häufig) andere Daten enthalten als die Apex-Seite im Browser. Dies ist zum Beispiel immer der Fall, wenn man ein normales Formular für eine Tabellenzeile einrichtet. Auf der Apex-Seite werden alle Daten einer Tabellenzeile in Eingabefeldern dargestellt, der Session State zeigt aber, dass alle Elemente dieser Seite leer sind, vielleicht mit Ausnahme einer Primärschlüsselspalte, die benutzt wird, um der Seite mitzuteilen, welche Zeile der Tabelle dargestellt werden soll. *Abbildung 1* zeigt, dass auf der Seite Daten zu sehen sind, der Session State aber leer ist.

Ist das nicht widersinnig? Wenn man eine Formularseite anzeigt, um eine Tabellenzeile zu editieren, müssten dann nicht initial im Session State und auf der

0.00000	0.00000	...Session State: Save "P5_ROWID" - saving same value: "AAAWbQAAKAAAADOAAG"	4
0.00000	0.00000	...Session State: Saved Item "P5_EMP_ID" New Value="106"	4
0.00000	0.00000	...Session State: Saved Item "P5_EMP_FIRST_NAME" New Value="Valli"	4
0.00000	0.00000	...Session State: Saved Item "P5_EMP_LAST_NAME" New Value="Pataballa"	4
0.00000	0.00000	...Session State: Saved Item "P5_EMP_EMAIL" New Value="VPATABAL"	4
0.00000	0.00000	...Session State: Saved Item "P5_EMP_PHONE_NUMBER" New Value="590.423.4560"	4
0.00000	0.00000	...Session State: Saved Item "P5_EMP_HIRE_DATE" New Value="05.02.06"	4
0.00000	0.00000	...Session State: Saved Item "P5_EMP_JOB_ID" New Value="IT_PROG"	4
0.00000	0.00000	...Session State: Saved Item "P5_EMP_SALARY" New Value="4900"	4
0.00000	0.00000	...Session State: Saved Item "P5_EMP_COMMISSION_PCT" New Value=""	4
0.00000	0.00000	...Session State: Saved Item "P5_EMP_EMP_ID" New Value="103"	4
0.00000	0.00000	...Session State: Saved Item "P5_EMP_DEP_ID" New Value="60"	4
0.00000	0.00000	Processes - point: ON_SUBMIT_BEFORE_COMPUTATION	4

Abbildung 2: Der Ablauf

Seite die gleichen Werte enthalten sein? Verständlich wäre eine Diskrepanz, wenn ein Feld auf der Seite verändert würde; wenn das jedoch nicht geschieht, warum sind dann Session State und Anwendungsseite nicht synchron?

Die Antwort lautet, dass Apex den Session State in diesem Fall nicht pflegen muss. Apex könnte dies tun, hätte aber nichts davon, denn die Daten werden aller Wahrscheinlichkeit nach durch das Formular verändert, daher ist der Session State sowieso sehr bald nicht mehr synchron. Und wenn die Seite abgeschickt wird, müssen die Eingaben des Benutzers ohnehin in den Session State übernommen werden; die vorher geleistete Arbeit wäre vergeblich gewesen. Daher wird beim normalen Darstellen einer Seite (im Response-Zyklus) der aktuelle Wert eines Formularfeldes aus der Workspace-Tabelle gelesen und im HTML der Seite integriert, aber nicht in den Session State kopiert. Eine Ausnahme ist, wenn die Daten eines Eingabefelds über eine Computation in das Element geschrieben werden.

Umgekehrt besteht die – so ziemlich – erste Aktivität beim Übermitteln einer Seite an Apex darin, die eingegebenen Werte der Formular-Elemente in den Session State zu kopieren. Man kann das beim Debuggen der Verarbeitung einer Seite sehen: Zuerst werden die „National Language Support"-Settings (NLS) gesetzt, anschließend die Session-ID auf Gültigkeit geprüft, und dann geht es los: Alle Elemente der Seite werden in den Session State kopiert. Die meisten Werte werden neu gesetzt (die Meldung lautet: „Saved Item "P5_EMP_ID" New Value="105"" anstatt „saving same value: "..."). Weil erst nach dieser Aktion die Computations, Validations, Processes und Branches ausgeführt werden, die für diese Seite definiert wurden, können diese auf dem Server auf den aktuellen Stand der Eingabefelder über den Session State zugreifen *(siehe Abbildung 2)*.

Probleme mit dem Session State

Es gehört zu den großen Vorteilen von Apex, dass die Interna der Session-State-Verwaltung nur ganz selten direkt programmiert werden müssen. Ein prominentes Beispiel, das einen in direkten Kontakt damit bringt, ist die Verwendung des Collection-API, das man zum Beispiel verwendet, um innerhalb eines Assistenten ein Tabular-Form einzusetzen. Das ist schon etwas fortgeschrittenes Arbeiten mit Apex, kommt aber natürlich vor.

Unabhängig davon sind Probleme in der Entwicklung von Apex-Seiten, die durch den Session State verursacht werden, allerdings an der Tagesordnung und normalerweise darin begründet, dass man nicht durchblickt, wann welche Daten im Session State enthalten sind und wann nicht.

Diese Fehler tauchen auf, wenn man den einfachen Request-Response-Pfad, der für Apex typisch ist, verlässt, zum Beispiel wenn ein Partial Refresh auf einen Bericht oder eine Region ausgeführt werden soll. Das Problem: Nach dem Response der Datenbank zeigt der Browser die aktuelle Seite. Für den Server ist „nach dem Response" nicht „vor dem Request", denn der Server erwartet nichts vom Client, sondern reagiert erst, wenn der nächste Request kommt. Nur im Zuge eines Request schreibt Apex automatisiert die Werte der Oberfläche in den Session State. Ansonsten hält der Session State still und enthält, je nachdem, entweder

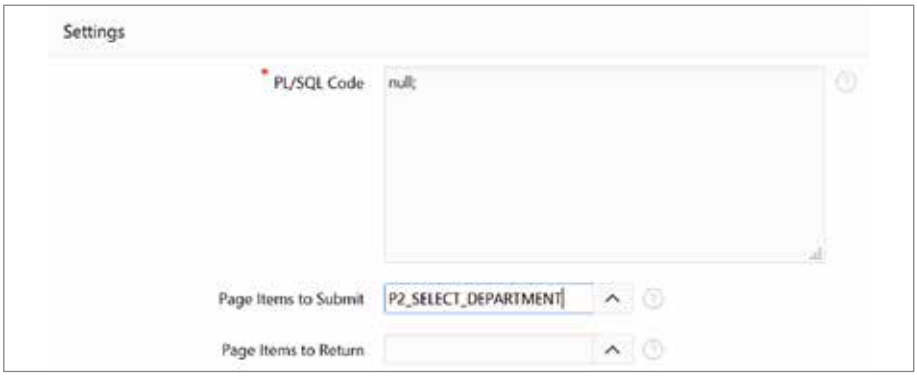

Abbildung 3: PL/SQL-Code „NULL;"

den initial gültigen Wert des Zeitpunkts, an dem die Seite ausgeliefert wurde, oder einfach „NULL“. Ein Partial Refresh durchbricht nun das Request-Response-Spiel, denn es wird nicht die Seite als Ganzes abgeschickt, sondern lediglich ein AJAX-Request abgesetzt. Dieser Request ist eine einfache Anfrage an den Server (die normalerweise nicht die Elementwerte auf der Seite umfasst), die vom Server auch beantwortet wird, ohne dass der Browser die Antwort als neue Seite aufbaut. Die Antwort wird vom AJAX-Request entgegengenommen und dynamisch in die bestehende Seite integriert. So lässt sich also zum Beispiel ein Bericht einer Seite aktualisieren, ohne alle anderen Seitenelemente erneut vom Server anfordern und durch den Browser darstellen lassen zu müssen.

Ein Szenario

Problematisch ist es, wenn ein Bericht auf einer SQL-Abfrage beruht, die durch einen Wert des Session State gefiltert wird. Man stelle sich eine Seite mit einer Auswahlliste und einen zugehörigen Bericht vor. Die Auswahlliste wählt die Abteilung, für die ein Bericht angezeigt werden soll. Als die Seite geladen wurde, war noch keine Abteilungsauswahl vorhanden; der Elementwert der Auswahlliste ist im Session State „NULL“. Damit durch die Auswahl einer Abteilung nicht die gesamte Seite neu geladen wird, wurde festgelegt, dass im Fall einer Änderung der Auswahlliste nichts weiter geschehen soll, damit man durch eine dynamische Aktion nur den Bericht aktualisieren kann. Diese dynamische Aktion ist als Refresh-Aktion auf die Berichtsregion angelegt, die immer dann feuert, wenn sich die Auswahlliste ändert.

Nun wird ein Eintrag der Auswahlliste gewählt. Dies geschieht im Browser, der Server erfährt davon nichts – der Session State bleibt daher unverändert. Der Bericht wird durch die dynamische Aktion aktualisiert, zeigt jedoch keine Daten. Im Licht des bisher Erklärten wird klar, warum: Der Session State enthält für das angegebene Element immer noch keinen Wert, die Filterung über einen fehlenden Wert hat zur Folge, das eben auch keine Ergebnisse gezeigt werden können.

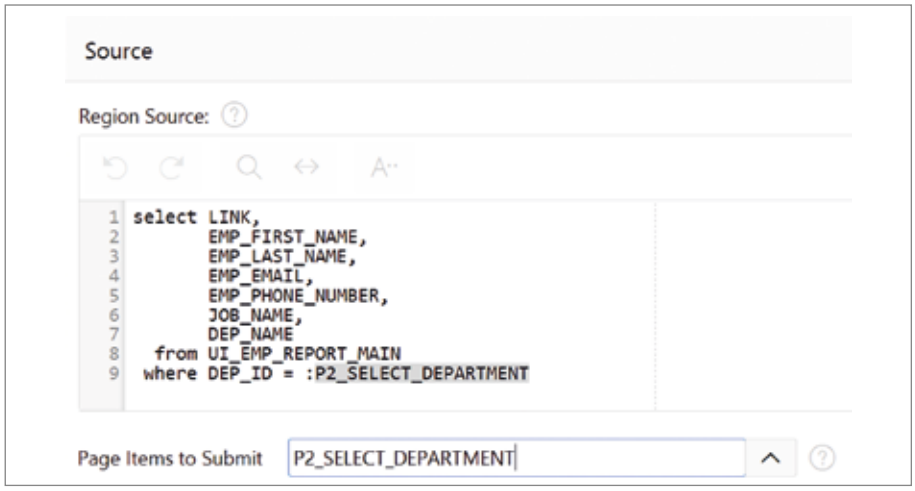

Abbildung 4: Eintrag des Seitenelements

Lösung des Problems

Eine erste Lösung des Problems besteht darin, der dynamischen Aktion eine zweite Aktion vom Typ „PL/SQL-Code ausführen“ hinzuzufügen. Der Prozess selbst soll in der Datenbank gar nichts tun, daher ist der Code, der ausgeführt wird, schlicht „NULL;“, aber der Dialog bietet die Möglichkeit, Elementwerte der Seite in den Session State zu übernehmen oder aus diesem zu lesen. Daher wird hier der Name der Auswahlliste in die Liste der Elementnamen, die an die Datenbank übermittelt werden sollen, eingetragen *(siehe Abbildung 3)*.

Es gilt jetzt noch darauf zu achten, dass diese zweite dynamische Aktion vor der Refresh-Aktion des Berichts ausgeführt wird (über die Sequenz-Nummer der Aktivitäten), sonst ist der Session State zu spät aktualisiert worden, als dass die SQL-Abfrage auf den Wert hätte zugreifen können.

Einfacher ist es, in den Einstellungen der Region einzutragen, welche Seitenelemente im Fall eines Refresh der Region übermittelt werden sollen. Diese Option verbirgt sich unterhalb der „select“-Anweisung für den Bericht *(siehe Abbildung 4)*.

Regionstypen, die keine dynamischen Aktualisierungen unterstützen, verfügen auch nicht über ein entsprechendes Eingabefeld. In diesem Fall werden die hier eingetragenen Seitenelemente automatisiert vor dem Aktualisieren des Berichts im Session State eingetragen, eine zusätzliche PL/SQL-Aktion ist daher nicht nötig. Das Problem lässt sich also dadurch lösen, dass dem Session State ein Wert übergeben wird, bevor eine Aktivität in der Datenbank ausgeführt wird.

Diese Problemlösung kann als Blaupause für ähnliche Probleme verwendet werden, beinahe immer ist die fehlende Harmonisierung des Session State Grund für unvorhergesehenes Verhalten von Apex bei dynamischen Aktivitäten. Nun gibt es den Einwand, dass diese Implementierung sehr unschön sei, sie zwei AJAX-Requests erfordere und immer auf diese Weise programmiert werden müsse. Das stimmt, allerdings mag sich dies in einer neuen Version von Apex verbessern (hoffen wir auf Version 5.1) und zum anderen ändert es nichts daran: Wesentlich ist zu verstehen, worin das Problem besteht. Die eleganteste Lösungsstrategie kann sich ändern, aber das Problem muss gelöst werden.

Zugriffsvarianten

Um auf die Werte des Session State zuzugreifen, könnte man theoretisch SQL einsetzen, allerdings ist dieser Weg nicht möglich, weil Sicherheitsfeatures von Apex dies nicht zulassen. Stattdessen bietet Apex verschiedene Möglichkeiten an, auf den aktuellen Session State zuzugreifen:

- Über eine Bindevariable mit dem gleichen Namen des Elements
- Über eine Funktion mit dem Namen „V“, der im gleichen Schema definiert ist und öffentlich über ein gleichnamiges Alias genutzt werden kann

Natürlich liefern alle Zugriffspfade das gleiche Ergebnis und alle liefern nur im Kontext einer Apex-Session überhaupt einen Wert. Ist ein Wert nicht vorhanden, wird übrigens kein Fehler, sondern lediglich „NULL“ geliefert, was einen dazu zwingt, akribisch auf korrekte Schreibweise zu achten. Hier könnte der Hinweis auf den Advisor in den Utilities hilfreich sein, denn dort werden Zugriffe auf Seitenelemente erkannt, die nicht existieren.

Zugriff auf die Bindevariablen

Gemäß den beiden oben geschilderten Zugriffspfaden stehen mehrere syntaktische Formen zur Verfügung, um den Session State abzufragen. Zunächst einmal lässt sich überall dort, wo PL/SQL eingesetzt wird, die Bindevariable über die Notation „:P1_ITEM“ ansprechen. Man kennt dieses Verfahren von Zeilen-Triggern, die ähnliche Umgebungsvariablen unter den Namen „NEW“ und „OLD“ anbieten. Immer dann, wenn der PL/SQL-Code selbst eine Variable nicht deklariert, sondern aus der Umgebung zur Verfügung gestellt bekommt, ist der vorangestellte Doppelpunkt zu verwenden.

Wird kein PL/SQL-Code ausgeführt, kann der Wert der Bindevariablen mit der Schreibweise „&P1_ITEM.“ angesprochen werden. Wichtig ist der abschließende Punkt hinter dem Namen des Elements. Er ist erforderlich, weil Apex dadurch erfährt, wo eine Ersetzungszeichenfolge endet. Nicht immer ist es möglich, das Ende durch ein Leerzeichen anzuzeigen, daher muss der Punkt diese Aufgabe übernehmen.

Auch diese Variante ist außerhalb von Apex bekannt, denn auf diese Weise wird in SQL*Plus eine Variable, die dort definiert wurde, in SQL oder PL/SQL als Ersetzungszeichenfolge eingefügt. Dieses Verfahren beschreibt bereits einen wesentlichen Unterschied zwischen den beiden syntaktischen Varianten: Die erste Variante („:P1_ITEM“) gilt als immun gegen SQL-Injection-Angriffe, weil die Variable typsicher ist und kein dynamisches SQL oder PL/SQL erfordert.

Im Gegensatz dazu ist die Verwendung der zweiten Variante einem Einsetzen einer Zeichenkette in eine dynamische SQL-Anweisung gleichzusetzen. Weil hierdurch potenziell die SQL-Anweisung selbst geändert werden kann, gilt dieser Ansatz als unsicher. Da aber nicht überall in Apex PL/SQL im Hintergrund verwendet wird, um eine Funktion auszuführen, kommt man manchmal nicht umhin, die Notation „&P1_ITEM.“ zu verwenden. Diese Schreibweise ist zum Beispiel erforderlich, wenn die Bezeichnung einer Region aus dem Wert eines Seitenelements abgeleitet werden soll, oder bei der Übergabe von Elementwerten als Parameter in Links.

Zugriff über die Funktion „V“

Wer möchte, kann immer dann, wenn PL/SQL im Einsatz ist, auch explizit die Funktion „V“ aufrufen und dadurch den Wert des Session State erfragen. Diese Variante bedeutet zunächst einmal mehr Schreibaufwand, schwerer wiegt jedoch die in Blogs häufig zu findende Empfehlung, diese Schreibweise nicht zu verwenden, weil sie langsam sei. Bevor darauf eingegangen wird, zunächst der Hinweis, dass es Situationen gibt, in denen die beiden anderen Schreibwei-

```
SQL> declare
  2      l_user_name emp.ename%type := 'BLAKE';
  3      l_deptno emp.deptno%type;
  4  begin
  5      select deptno
  6          into l_deptno
  7          from emp
  8      where ename = l_user_name;
  9  end;
  10 /

PL/SQL-Prozedur erfolgreich abgeschlossen.
```

Listing 1

```
SQL> select ename, dbms_random.value zufallszahl
  2  from emp;

ENAME         ZUFALLSZAHL
----------    -----------
SMITH         ,689783172
ALLEN         ,512884812
WARD          ,710363434
...
14 Zeilen ausgewählt
```

Listing 2

```
SQL> select ename,
  2      (select dbms_random.value
  3          from dual) zufallszahl
  4  from emp;

ENAME         ZUFALLSZAHL
----------    -----------
SMITH         ,664207243
ALLEN         ,664207243
WARD          ,664207243
JONES         ,664207243
...
14 Zeilen ausgewählt
```

Listing 3

sen nicht funktionieren und von daher nur dieser Weg bleibt, um mit dem Session State zu interagieren: in PL/SQL-Packages und Views zum Beispiel, also immer außerhalb der Apex-Seiten selbst. Nun liest man genauso häufig die Empfehlung, kein ausuferndes PL/SQL oder SQL auf den Apex-Seiten selbst zu integrieren, sondern stattdessen diesen Code als Package oder View in die Datenbank auszulagern. Ein Widerspruch also, der nach einer Lösung verlangt.

```
SQL>  with params as
  2              (select dbms_random.value zufallszahl
  3                 from dual)
  4   select ename,
  5              zufallszahl
  6          from emp
  7       cross join params;

ENAME         ZUFALLSZAHL
----------    -----------
SMITH         ,481366804
ALLEN         ,692801788
WARD          ,740467425
JONES         ,701446446
...
14 Zeilen ausgewählt
```

Listing 4

```
SQL>  with params as
  2          (select /*+ outline */
  3                  dbms_random.value zufallszahl
  4              from dual)
  5   select ename,
  6          zufallszahl
  7          from emp
  8       cross join params;

ENAME         ZUFALLSZAHL
----------    -----------
SMITH         ,212091558
ALLEN         ,672092931
WARD          ,672092931
JONES         ,672092931
...
14 Zeilen ausgewählt
```

Listing 5

```
SQL> declare
  2       cursor emp_cur is
  3           select ename, job, sal
  4               from emp;
  5   begin
  6       for emp in emp_cur loop
  7           dbms_output.put_line(
  8               emp.ename || ' erhaelt die ' ||
  9               v('P1_AUSZEICHNUNG') ||
  10              ' fuer vorbildlichen PL/SQL-Code');
  11      end loop;
  12  end;
  13  /
```

Listing 6

Zunächst eine Erläuterung des Problems: Warum sollte der Zugriff auf „V“ langsamer sein als der Zugriff auf eine Binde-Variable in der „:“-Notation? Die Implementierung von PL/SQL sorgt dafür, dass Variablen, die in SQL eingesetzt werden, vorab als konstante Zeichenfolgen in die SQL-Anweisung integriert werden, bevor diese ausgeführt wird.

In *Listing 1* wird der Wert „BLAKE“ als Konstante in die „select“-Anweisung integriert und diese erst dann ausgeführt. Dadurch kann SQL effizient arbeiten. Anders ist das, wenn eine PL/SQL-Funktion ins Spiel kommt, wie etwa die Funktion „V“. PL/SQL (und SQL schon gleich gar nicht) kann nicht voraussehen, welches Ergebnis „V“ für einen Parameter liefern wird. Das Ergebnis hängt ja vom Session State ab, der für jede Session und auch innerhalb einer Session über die Zeit verschieden sein kann. Daher kann der Compiler das Ergebnis der Funktion nicht einfach durch sein Ergebnis ersetzen, auch wenn das in diesem konkreten Fall logisch möglich wäre.

In *Listing 2* erwartet man nun für jede Zeile der Ergebnismenge eine neue Zufallszahl. Hätte die Datenbank einfach einmal eine Zufallszahl berechnet und diese dann ausgegeben, wäre der Sinn der Funktion verfehlt. Daher muss also eine Funktion wie „V“ in SQL stets berechnet werden. Diese Berechnung erfolgt aber in PL/SQL, sodass SQL für jede betroffene Zeile einen Umgebungswechsel zwischen SQL und PL/SQL durchführen muss. Schlimmer noch: Die Funktion „V“ ermittelt den Wert des Session State durch eine SQL-Abfrage gegen die Tabelle „WWV_FLOW_DATA“, macht also seinerseits einen Umgebungswechsel nach SQL. Diese Umgebungswechsel sind sehr aufwändig und summieren sich auf erhebliche Werte, wenn sie sehr oft erfolgen.

Die Verwendung der Funktion „V“ in der „where“-Klausel einer SQL-Anweisung ist also fatal, denn dann muss für jede Zeile der Tabelle die Funktion berechnet werden, selbst wenn nur eine Zeile als Ergebnis der Filterung übrig bliebe. Dies ist im Zusammenhang mit der Funktion „V“ besonders ärgerlich, da sie ihren Wert gar nicht ändern kann, denn dazu müsste der Anwender in seiner Session ja den Session State ändern,

und das kann er während einer „select"-Abfrage nicht.

Problemlösung und Empfehlungen

So lässt sich beides erreichen: Code in die Datenbank verlegen und dennoch eine performante Anwendung erhalten. Die Lösung des Problems, der Umgebungswechsel in SQL, ist immer die gleiche, obschon sie zunächst überraschend aussieht: Man schreibt den Funktionsaufruf in eine skalare Unterabfrage der Form „where ename = (select v('P1_ENAME') from dual)". Der Effekt lässt sich schön durch den Aufruf der Funktion „dbms_random" aus dem vorigen Beispiel erkennen *(siehe Listing 3)*.

SQL wird eine skalare Unterabfrage vorab einmal berechnen und das Ergebnis als Konstante für die restliche „select"-Anweisung betrachten: Die negativen Folgen des Umgebungswechsels bestehen nicht mehr. Hat man mehrere Seitenelemente, die man für eine SQL-Anweisung benötigt, könnte man geneigt sein, diese in einer Unterabfrage zu sammeln und anschließend gegen diese Unterabfrage zu joinen. Das ist grundsätzlich eine gute Idee, allerdings ist der Optimizer hier ein wenig übereifrig: Er meint zu erkennen, dass diese Unterabfrage gar nicht nötig ist, und löst sie wieder auf, was dazu führt, dass alle Funktionen wieder für jede Zeile aufgerufen werden.

Zumindest in Datenbank-Version 12*c* lässt sich dies durch den Optimizer-Hint „/*+ outline */" in der Unterabfrage verhindern (interessant, dass die Datenbank in diesem Fall die Funktion zweimal aufruft), ansonsten bleibt nur der Weg, jeweils eine separate skalare Unterabfrage für jeden Session-State-Wert zu verwenden *(siehe Listing 4 und 5)*.

```
SQL> declare
  2      cursor emp_cur is
  3          select ename, job, sal
  4              from emp;
  5      l_auszeichnung varchar2(200);
  6  begin
  7      l_auszeichnung := v('P1_AUSZEICHNUNG');
  8      for emp in emp_cur loop
  9          dbms_output.put_line(
 10              emp.ename || ' erhaelt die ' ||
 11              l_auszeichnung ||
 12              ' fuer vorbildlichen PL/SQL-Code');
 13      end loop;
 14  end;
 15  /
```

Listing 7

Die Funktion „V" in PL/SQL verwenden

Analog lautet die Empfehlung für PL/SQL, den Aufruf der Funktion „V" nicht in einer Schleife in PL/SQL einzubauen, sondern die benötigten Werte aus dem Session State vor dem Aufruf der Schleife in lokale Variablen umzukopieren, denn auch in einer Schleife wären sonst unnötige Umgebungswechsel zwischen PL/SQL und SQL unvermeidlich. *Listing 6* würde man also besser wie in *Listing 7* formulieren.

Aber auch dieser Rat gilt generell: Liefert ein Funktionsaufruf ein konstantes Ergebnis, ist er wie eine konstante Variablen-Zuordnung vor einer Schleife und nicht in ihr durchzuführen. Besondere Beachtung sollten hier auch die Funktionen „SYSDATE" und vor allem „USER" erfahren, da diese auch Umgebungswechsel zur Folge haben können beziehungsweise bei „USER" in jedem Fall haben werden.

Empfehlung

Apex erfordert, sich in die Denkweise des Session State einzufinden und sich genau klarzumachen, was in welcher Reihenfolge bei der Verarbeitung einer Anfrage in Apex geschieht. Nur, wer sich in dieser Arbeitsweise auskennt, wird Fehler vermeiden können. Der Autor erinnert sich an diese Fehler aus seiner eigenen Einarbeitungsphase nur zu gut und ist fest davon überzeugt, dass die Kenntnis des Session State und dessen Arbeitsweise zu den Kern-Kompetenzen gehört, um mit Apex erfolgreich arbeiten zu können. Wer sich die Vor- und Nachteile der unterschiedlichen Zugriffsstrategien auf den Session State zu eigen macht, ist in der Lage, Sicherheitsprobleme zu vermeiden und gleichzeitig aufgeräumten, lesbaren und schnellen Code zu erstellen.

Jürgen Sieben
j.sieben@condes.de

Kritische Lücke in Java SE: Oracle veröffentlicht außerplanmäßiges Update

DOAG Online

Oracle hat ein Patch für Java SE bereitgestellt, das eine kritische Sicherheitslücke schließen soll. Der Fehler erlaubt es Angreifern, ein System vollständig zu übernehmen. Betroffen sind Oracles Java SE 7 Update 97 und Version 8 Update 73 und 74 für Windows, Mac OS X, Linux und Solaris.

Nach Angaben von Oracle können Angreifer die Lücke dazu nutzen, um über eine speziell präparierte Webseite Schadcode aus der Ferne und ohne Authentifizierung einzuschleusen und auszuführen. Die Schwachstelle ist auf der Skala des umfassenden „Common Vulnerability Scoring System" mit 9,3 Punkten bewertet, wobei 10 die höchste Stufe darstellt.

Da die technischen Details der Schwachstelle bereits veröffentlicht wurden, empfiehlt Oracle allen Nutzern dringend zu einem Update.